Congestion Aware Priority Flow Control in Data Center Networks

Serhat Nazim Avci, Zhenjiang Li, Fangping Liu
Futurewei Technologies
{serhat.avci,andy.z.li,julius.f.liu}@huawei.com

***Abstract*—The data center bridging (DCB) protocols enable Ethernet to become the leading unified fabric for a converged data center. As a part of DCB, priority flow control (PFC) is a control mechanism that provides the losslessness feature required by certain applications such as Fibre Channel over Ethernet. Quantized congestion notification (QCN) is also a DCB protocol and provides congestion control functionality to complement PFC. However, in today's practical data center networks, QCN is not feasible and PFC is not efficient against congestion. We propose a new link layer mechanism called congestion aware priority flow control (CaPFC) that equips PFC with better congestion control capability in the absence of QCN. Changes brought by CaPFC on top of PFC are lightweight. By conducting simulations in NS-3, we demonstrate that CaPFC handles persistent and flash congestion much better than PFC. For the short messaging traffic and the query traffic, which are sensitive to tail latencies, CaPFC consistently outperforms PFC in terms of tail flow completion time.**

I. Introduction

One of the major trends in data center networking is the fabric convergence, which helps to eliminate separate networks for servers, storage and computing, thereby reducing the costs related to maintenance, management, training, equipment, cabling, power and space [1]. Data center bridging (DCB) is an effort by the IEEE 802.1 working group to promote Ethernet as the unified fabric of data center networks by expanding its networking and management capabilities, which is also called as Converged Enhanced Ethernet. DCB requires per-priority link level flow control, traffic differentiation, congestion management and transmission scheduling features to make Ethernet a candidate for unified data center networking fabric [2]. For that purpose, DCB includes four different protocols, namely priority-based flow control (PFC), enhanced transmission selection (ETS), congestion notification (QCN) and data center bridging exchange protocol.

Although the other two protocols are also important to DCB, only PFC and QCN play a role in the congestion control of the flows, which is the focus of this paper. The goal of PFC is to push the congestion from inside the network to the edges by recursively propagating this congestion to upstream nodes. In theory, PFC is envisioned to keep flow control for different priorities independent and hence create virtual lanes. Congestion created by one priority pauses only the traffic of that specific priority and does not affect the traffic of other priorities. QCN is developed to carry out large scale persistent congestion management in Ethernet-based data center networks.

PFC and QCN have proved to be successful to a certain extent. However, they have practical issues that limit their deployment. In one hand, even though PFC manages to prevent packet losses under certain assumptions, it has certain drawbacks such as congestion spread, head of the line (HOL) blocking,and potential deadlocks.

In another hand, despite the necessity of a congestion control mechanism that will complement PFC in datacenter networks and the proven performance of QCN in testbeds, QCN has serious issues that prevent its wide deployment. First of all, the scope of QCN is restricted to a single VLAN. It cannot pass through network borders [3] in data centers or Fibre Channel Forwarders (FCF) in Fibre Channel over Ethernet networks [4]. Second, QCN increases the switch and adapter complexity [2]. In a DCB switch, hundreds of congestion points are needed in addition to the tens to thousands rate limiter units in the adapters. They put a burden on the network devices in terms of complexity, power and cost. Third, QCN lacks application control, which is also pointed by [2]. As increasingly more applications prefer to control the rate at end-hosts, the QCN reaction point is disabled by default in order to prevent conflict with the rate control of applications.

This paper has three main contributions.

- We show how some overlooked assumptions about PFC fail in most merchant silicon data center switches. We show how PFC fails to prevent packet drops between ingress and egress queues in a typical merchant silicon switch. When trying to fix this issue with the current architecture of PFC, we also demonstrate that backpressure caused by PFC results in congestion spread not only among different ports and switches but also within ideally independent priorities.
- We propose congestion aware priority flow control (CaPFC) as a novel link layer mechanism that incorporates congestion control functionality in PFC. CaPFC is designed to address the shortcomings of PFC in the absence QCN. It employs a joint ingress and egress queue monitoring mechanism to handle congestion proactively.
- We demonstrate that two versions of CaPFC consistently outperforms two versions of PFC in terms of tail flow completion time. For that purpose, we build a simulation setup in NS-3 to evaluate the performance of the proposed CaPFC compared to PFC.

In the next section, we provide the technical details on the research problem and describe the proposed CaPFC mechanism. Next, we discuss how to tune the parameters in CaPFC

ISBN 978-3-901882-83-8

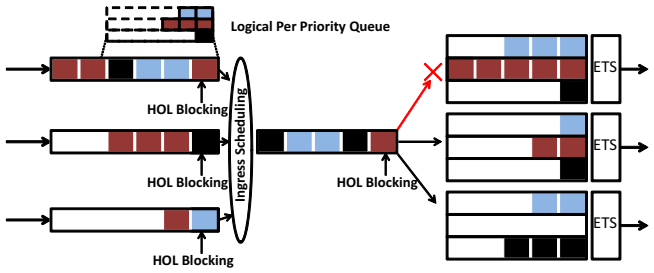

Fig. 1. The switch architecture and the HOL blocking

in Section III. The simulation setup and the simulation results are presented in Section IV. In Section V, a brief summary of the related work is presented before concluding the paper.

II. Congestion Aware Priority Flow Control

In this section, we present the proposed CaPFC mechanism and how it is integrated into the legacy PFC architecture. Before that, a typical merchant silicon architecture is presented to define the research problem and the motivation.

A. Problem Definition

The architecture of the switches that are commonly deployed in today's data center networks has a critical impact on the design and performance of the flow and congestion control mechanisms. One of the key insights behind this paper is the inefficient propagation of congestion from the egress ports to the ingress ports of a network device. The way PFC messages propagated from inside the network to the edge hosts is not very clearly analyzed. The usual assumption is that when an egress buffer of a paused port gets filled up, the packets starts to fill ingress buffers, hence PFC messages are created to the upstream device. However, there is no clear mechanism in PFC standards on how to stop switching packets from ingress to egress if there is no space left at the egress.

The reference merchant silicon switch architecture has a pipelined packet processor and a traffic manager. Whenever a packet is admitted from an ingress port, first, its header is extracted. The header is sent to a pipeline to decide what to do with the packet while the payload is stored in the ingress buffers or in shared memory buffers depending on the type of the switch. At the ingress ports, even though packets are processed on a FIFO order, we logically implement a separate queue per priority by keeping per-priority packet counters. PFC generates pause (XOFF) and unpause (XON) messages per-priority with respect to counter of each priority. When admitting packets to the pipeline, round robin scheduling is implemented between different input ports.

Unlike crossbar type of architectures, the packets are put into the egress queue whenever a forwarding decision is made, which is only after the packet processing phase. Therefore, once a forwarding decision is made for a packet, it will either will be placed at the corresponding egress queue if there is any empty space or it will be dropped. This has a sinister effect on the data transmission between ports when one of the egress queues for a single priority is completely full. The only way to prevent packet drop is to stop packet processing which will halt the data transmission towards any egress port for any priority. In this case, the packet belonging to other priorities or designated to different ports will suffer from the HOL blocking as shown in Fig. 1. In most of the merchant silicon switches, it is not feasible to stop packet processing [5] which means PFC cannot completely prevent packet drops. Dropping packets results in lengthy timeouts and retransmission and aggravates the latency and the congestion in the network. In Section IV, congestion spread, packet drops, and HOL among different priorities are empirically observed to cause high tail latencies for short flows.

This issue exists also in shared memory merchant silicon switches. In these switches, the payload is buffered only once, therefore moving a packet from ingress to egress does not occupy extra buffer space. However, the header of the packet is queued at the corresponding egress port when a decision is made. In most merchant silicon switches, the egress queues have queuing and scheduling policies to satisfy the quality of service which limit the capacity of the queues [5]. When an egress queue reaches that limit, a packet is dropped from ingress to egress even in shared memory switches.

Even though PFC is envisioned to keep flow control for different priorities independent and hence create virtual lanes, in typical switches, egress buffer overflow in one priority causes either HOL at all ingress ports for every priority or results in packet drops at the end of packet processing. Eventually, it causes a saturation tree inside the network in less than a few round-trip times, [6], which makes software solutions too slow to succeed forcing the adoption of hardware solutions. Both of these scenarios dramatically increase the tail completion time of short and time-sensitive flows. To prevent these scenarios, the key idea behind CaPFC is to develop a proactive congestion aware flow control mechanism which will keep the buffers on the egress ports from being completely occupied. In CaPFC, the virtual lanes for each priority are preserved by preventing congestion in one priority spreading to other priorities.

B. CaPFC Introduction

In typical PFC, ingress queue states are taken into account when deciding the state of flow control. CaPFC also monitors egress queues in order to add congestion control awareness on top of PFC. CaPFC employs per-input-port counters at each egress buffer to measure the congestion contribution of each input port to that specific egress buffer. It is important that these measurements are fresh and do not increase complexity as we describe in Section II-D.

In Fig. 2, it is shown that different ingress queues may send packets to a single egress queue of the same priority. If the buffer occupancy of an egress queue for a specific priority passes a certain threshold, the queue starts to count the number of packets newly arriving from each input interface. If the buffer occupancy passes the congestion threshold (explained in Section II-D), CaPFC finds the highest contributor or highest contributors of that congestion among ingress interfaces. After identifying those culprit ingress interfaces, the egress queue ask those ingress interfaces to issue PFC-XOFF messages to their corresponding upstream devices. It should be noted that CaPFC never pauses ingress scheduling for any ingress queue instead it asks them to pause the traffic from their upstream devices. The details of the CaPFC operation is explained in the following subsections.

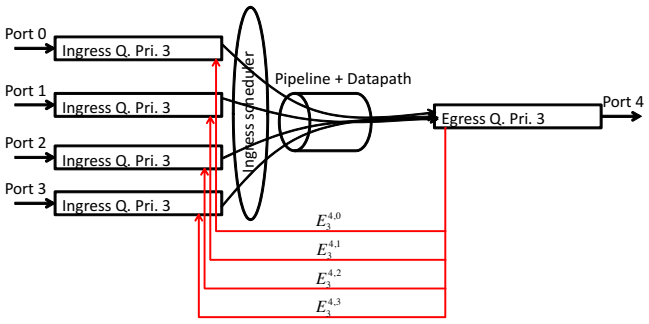

Fig. 2. An egress queue of a priority receives packets from ingress queues of other ports with that priority. In return, it sends congestion state signals per-ingress queue per-priority.

C. Ingress Congestion Management

CaPFC ingress queue management is the same as it is in PFC. CaPFC monitors the buffer occupancy of ingress queues which are PFC-enabled. Per-priority ingress buffer state keeps track of the ingress congestion behaviour. Buffer occupancy over time of an ingress queue of port i for priority p is B_p^i and it is shown in Fig. 3. MAX threshold is the maximum capacity of the queue. The variable ${I_p}^i$ keeps ingress queue congestion state of port i for priority p. It is equal to 1 in congestion (XOFF) state and 0 in non-congestion (XON) state. In the beginning, XON is the default state. In Fig. 3, when the buffer occupancy passes the XOFF threshold, this queue switches to XOFF state which is shown by dashed red line. The queue turns back to XON state once the buffer occupancy drops below the XON threshold.

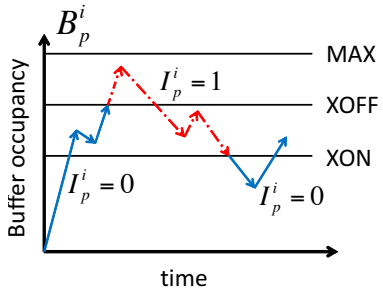

Fig. 3. Buffer occupancy over time for ingress queue buffers.

D. Egress Congestion Management

Buffer occupancy of the egress queues are monitored as in QCN. The goal is to detect congestion in the egress queues and identify the heavy flows that contribute most to the congestion. In Fig. 4, the buffer occupancy over time graph of an egress queue is depicted. In addition to the MAX, XOFF, and XON thresholds, the WARN threshold is added. As in the ingress queues, XOFF and XON thresholds are used to detect congestion and the lack thereof. MAX threshold is the absolute capacity of the buffer. If it is reached, the new packets are dropped. Before explaining the WARN threshold, we introduce a new variable $C_p^{i,o}$ which keeps the arrival count of the packets in egress queue of port o of priority p from input port i. These counters are used to identify the heaviest contributors to the congestion at the egress queues. However, they bring additional complexity since they need to be updated every time a new packet comes in to an egress queue. In addition, the freshness of these counters is also important to accurately identify the congestion contributors. Therefore, CaPFC introduces the WARN threshold to trigger those counters only in the congestion state. If the buffer occupancy is below the WARN threshold, the counters are reset to 0. When the buffer occupancy passes the WARN threshold, the counters are activated to keep the number of packets arriving from each ingress port for a specific egress queue. WARN threshold helps to reduce complexity and keep counters fresh and accurate. In Section III, we discuss how to tune these thresholds to optimize the performance.

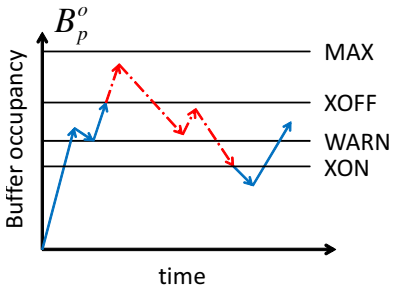

Fig. 4. Buffer occupancy over time for egress queue buffers.

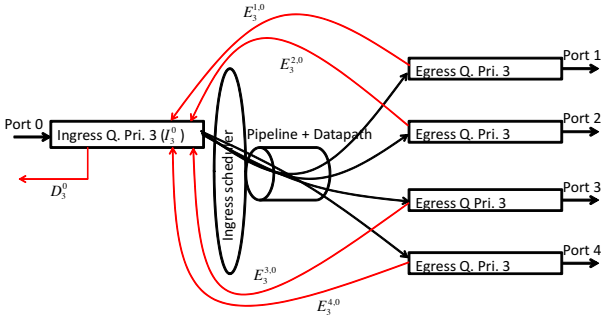

Fig. 5. Packets from a single ingress queue are forwarded to all egress queues per-priority. In return, the ingress queue receives a congestion state signal from each egress queue per-priority.

CaPFC introduces the variable $E_p^{i,o}$ to keep the congestion state of an egress port o for input port i and for priority queue p. These congestion state values are fed back to the corresponding input ports as shown in Fig. 5. These state variables are then used to determine to send XON or XOFF messages out of those input ports to the upstream devices. In CaPFC, egress queues do not pause the traffic coming out of ingress queues to avoid HOL which helps to preserve virtual parallel lanes for each priority.

Stop-Max is the name of the first algorithm that is used to calculate the congestion states where only the maximum contributor of the congestion is identified and the egress congestion signal $E_p^{i,o}$ is set to 1 only for that input port. If the congestion persists, the counter of other input ports continue to increase and the max contributor index changes. At that point, the new maximum counter input port is also put in congestion state and asked to pause upstream traffic. Input ports are put into congestion state one-by-one as long as congestion persists. The pseudocode of this algorithm is given in Algorithm 1.

The second algorithm is named Stop-Calibrate Policy. When the buffer occupancy is above the XOFF threshold, it

Algorithm 1 Stop-Max Algorithm

Require: $B_p^{o,+} = B_p^o$ + size of the new packet
Require: $B_p^{o,-} = B_p^o$ - size of the dequeued packet
1: **if** Enqueue a packet at port o with priority p **then**
2: **if** $B_p^{o,+} \geq MAX$ threshold **then**
3: Drop the packet
4: **else**
5: Accept the packet
6: $B_p^o \leftarrow B_p^{o,+}$
7: **if** $B_p^o \geq WARN$ threshold **then**
8: $C_p^{i,o} \leftarrow C_p^{i,o} + 1$
9: **if** $B_p^o > XOFF$ threshold **then**
10: Detect the heaviest contributor i^*
11: $i^* \leftarrow \arg\max_i C_p^{i,o}$
12: Put it into congestion state
13: $E_p^{i^*,o} \leftarrow 1$
14: **else if** Dequeue a packet from port o with priority p **then**
15: Remove the packet
16: $B_p^o \leftarrow B_p^{o,-}$
17: **if** $B_p^o \leq WARN$ threshold **then**
18: $C_p^{i,o} \leftarrow 0 \quad \forall i$
19: **if** $B_p^o \leq XON$ threshold **then**
20: $E_p^{i,o} \leftarrow 0 \quad \forall i$

first sorts the congestion counters of the egress queue. Then starting from the highest, it sets the congestion signal for the input ports whose cumulative counter percentages pass a predefined cut-off threshold ratio. This threshold can be calibrated between 0 and 1 defining the aggressiveness of the CaPFC technique. If the threshold ratio is set as 1, then congestion signal of every input port with a packet inside the egress queue is set to 1. In Algorithm 2, we only reflect the differences from the Algorithm 1. The steps 10 and 13 in Algorithm 1 are replaced by the steps in Algorithm 2. CUT is the calibrated cut-off threshold, the ratio of the congestion contribution that is determined to be heavy.

Algorithm 2 Changes for Stop-Calibrate Policy

1: Detect the heaviest contributors ▷ In place of Step 10
2: Sort congestion counters $C_p^{i,o}$ based on i
3: Find the set of input ports IP whose cumulative congestion contribution ratio passes the CUT threshold
4: $E_p^{i,o} \leftarrow 1 \quad \forall i \in IP$ ▷ In place of Step 13

E. Combined Congestion Reaction

In PFC, if the ingress queue of a certain priority gets into a congestion state, it sends a PFC-XOFF message to pause the incoming traffic. CaPFC incorporates both ingress queue state and egress queue states to pause and unpause the incoming traffic. An ingress queue may send packets to different egress queues of the same priority as depicted in Fig. 5. Likewise, it receives an egress state signal $E_p^{i,o}$ destined for this input port. If any of the egress ports sees the traffic coming from ingress queue i is responsible for the congestion at egress queue o, they will notify the ingress queue with the $E_p^{i,o}$ signal set to 1. Therefore, the decision to send an XOFF or XON message to the upstream device is based on the congestion state of the ingress queues and the congestion state signals received from all egress queues. In input port i, for priority p, the combined congestion state variable D_p^i is calculated by

$$D_p^i = I_p^i \vee \bigvee_{\substack{o \in P, \\ o \neq i}} E_p^{i,o} \qquad \forall i \in P, 1 \leq p \leq 8,$$

where P is the set of ports. For example, if we assume the switch has five input/output ports indexed as $0, 1, 2, 3, 4$, then the CaPFC decision variable for port 0 for priority 3 will be equal to

$$D_3^0 = I_3^0 \vee E_3^{0,1} \vee E_3^{0,2} \vee E_3^{0,3} \vee E_3^{0,4}.$$

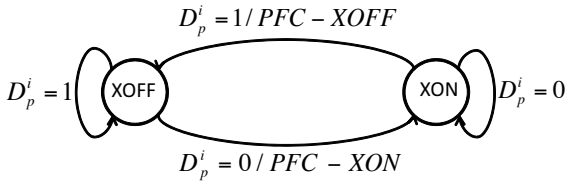

Fig. 6. The decision logic behind CaPFC, which is very similar to that of PFC except I_p^i is replaced by D_p^i.

In order to reduce complexity and overhead, we assume that congestion state signals E_p^i are only sent when there is a change of state. The ingress queue stores the latest updates of these signals and recalculates D_p^i only if at least one of the state signals change. The XOFF/XON behaviour of a receiving port based on variable D_p^i is highlighted with a finite state machine in Fig. 6. Compared to PFC, the only difference is the CaPFC messages are triggered by combined congestion state signals D_p^i instead of just ingress congestion state signals I_p^i. In short, the traffic incoming at port i with priority p is paused if the ingress queue is in congestion state or the ingress queue is deemed responsible for congestion at any of the egress queues, for that priority. The traffic for priority p is unpaused only if both the ingress queue is out of congestion state for that priority and is not responsible for the congestion in any of the egress queues for that priority.

F. Shared Memory Switches

In shared memory merchant silicon switches, every priority queue at every port has a small dedicated memory. The rest of the memory is assigned to a common pool. There are per-priority-queue, per-port, per-all-ingress-ports, per-all-egress-ports thresholds on the buffer usage from this shared memory pool. When admitting a packet from an ingress queue to an egress queue, the buffer occupancy counters of that egress queue, counters of that egress port and the total buffer occupancy of all egress ports are checked. If any of those counters are equal to the capacity threshold, this packet cannot be admitted. When an ingress queue admits a packet, it also checks the buffer occupancy counter of total shared memory usage in that switch. If there is no space left in the shared memory pool, the packet is not admitted. In our CaPFC implementation, setting XOFF, XON, and WARN thresholds for each buffer usage counter would be complex. Therefore, we implement CaPFC only on per-priority-queue buffer usage and set those thresholds with more safety margin to mitigate the buffer overflow with respect to other counters.

G. CaPFC Implementation Requirements

The goal behind CaPFC is to design a congestion aware flow control scheme with maximum added functionality but with minimum extra complexity. It is crucial that CaPFC is easily integrated to the legacy hardware in which PFC has already been deployed. As depicted in Fig. 7, CaPFC assumes no change on top PFC from a network perspective. The transmitter and receiver mechanisms of PFC is intact whereas the decision logic behind XOFF and XON messages is modified. Therefore, CaPFC does not require any change in the network interface cards (NICs) but requires monitoring of the egress queues, signaling between the ingress and egress queues and a modified logic in the ingress queues. All of these changes are easy to adapt in practical switches.

CaPFC introduces per-input-port per-priority counters at egress ports. CaPFC is envisioned to be used only for lossless traffic services. In a network, typically few of eight classes of services will use lossless services. Therefore, the counters for this purpose will be typically lower end of $O(P^2)$, where P is the number of ports. In popular commodity switches with 48 ports, that corresponds to maximum 18432 counters for 8 priorities, which is manageable in those switches. If each counter occupies 2 bytes of memory, it corresponds to approximately 36 KB extra memory, which is easily affordable. In high-radix switches with 128 ports, the CaPFC service would be enabled for a small number of lossless priorities, which keeps the number of total counters around 10K's.

The computational complexity brought by CaPFC is updating the congestion counters every time a new packet comes in and goes out of an egress queue and calculating the heaviest ingress ports. This complexity is significantly simplified after the introduction of the WARN threshold since only the lossless queues which observe high levels of buffer occupancies need to update their counters at the line rate. A software solution may not scale in the cases when multiple queues observe high occupancy levels, therefore a hardware solution is a better fit. A dedicated FPGA can keep the counters in a table per egress port and update them in line rate if the corresponding queue occupancy is over the WARN threshold. Once the occupancy of an egress queue passes the XOFF threshold, the FGPA module is responsible to sort the per-ingress-port counters, which has $O(PlogP)$ complexity. In Stop-Max algorithm, the congestion signal of the index of the maximum counter is set to 1. In Stop-Calibrate algorithm, the minimum set of the indexes whose counter contributions exceed the CUT threshold are found. It is carried out by cumulating the counters in the decreasing order until the threshold ratio is passed. It has a $O(P)$ complexity. Overall, CaPFC has a limited memory requirement and requires a small die space with $O(PlogP)$ computational complexity in an FPGA.

On the other hand, CaPFC mitigates the disadvantages of QCN. Unlike QCN, CaPFC can cross VLAN borders since it uses the same network interface of PFC as shown in Fig. 7. Second, it incurs much less complexity compared to QCN. In Stop-Max algorithm, it introduces only the WARN threshold. For simplicity, the WARN threshold is a virtual register which is represented by the XON threshold in the switch ASIC. In Stop-Calibrate algorithm, the cut-off parameter is also introduced. On the other hand, a typical QCN implementation introduces 12 extra parameters [7]. The tuning complexity of those parameters are also much higher. In addition, CaPFC works per-input port whereas QCN works per-flow, which also increases the complexity of the operation. Finally, unlike QCN, CaPFC does not require any change in the end-hosts. In conclusion, CaPFC adds significant congestion control functionality to flow control without introducing compatibility issues and significant complexity concerns.

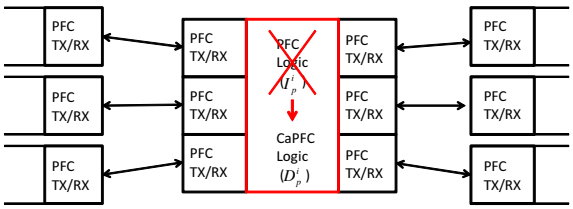

Fig. 7. CaPFC does not change the transceiver structure already deployed for PFC. CaPFC only replaces the decision logic of PFC by incorporating egress queue states along with the ingress queue state per-priority per-input port. The ingress decision variable I_p^i of PFC is replaced by combined decision variable $D_p^i = I_p^i \vee \bigvee_{o \in P,\ o \neq i} E_p^{i,o}$ of CaPFC.

H. Cross Layer Interaction

We believe a comprehensive solution should come from a multi-layer approach which integrates different techniques. Therefore, we built CaPFC on the cross-layer architecture of DeTail [8]. In DeTail, besides a lossless fabric and hierarchical prioritization, there are changes in the network layer and transport layer. In network layer, it proposes a per-packet adaptive load balancing scheme which dominates Flow Hashing and Lossless Packet Scatter in terms of reducing congestion hotspots [8]. In addition, the transport layer is modified to be reorder-resistant since packets can take different routes due to Adaptive Load Balancing. The transport layer of DeTail uses explicit congestion notification (ECN) protocol to detect congestion. CaPFC interacts with TCP in the same way PFC does as explained in [8]. CaPFC redesigns the link layer portion of DeTail but requires no change in the network and transport layers except tuning of the parameters.

III. Parameter Selection Analysis

Setting parameters in PFC is limited to finding XOFF and XON thresholds assuming the MAX threshold is given by the hardware specs. It is clear that XON must be smaller than XOFF. In selecting the XOFF threshold, the idea is to put the minimum amount of headroom buffer that will absorb all of the traffic that was received after a PFC pause message is sent to the upstream device [9]. To calculate this value, maximum transmission unit (MTU) on the transmitting end of receiver, length of the cable, speed of the wire, transceiver latency, response time of sender, and MTU on the transmitting end of sender are taken into account. To simplfiy, the headroom buffer must be approximately equal to maximum link capacity times RTT between two links. The XOFF threshold is set by subtracting the headroom buffer from the MAX threshold. Setting XON threshold is more challenging than the setting the XOFF threshold. Setting it too low causes buffer underflow, low link utilization and congestion spreading [8]. Setting it too high causes too many PFC messages which creates too much overhead in return [8]. We tuned the value of XON threshold according to the on simulation results.

In CaPFC, XOFF and XON thresholds of the ingress buffers are set the same as in PFC. At the egress buffers, setting XOFF threshold requires a deeper analysis. Even if the egress queue sends congestion state signals to trigger a pause message at all input ports, it needs to take the packets already in the ingress buffers and on the links into account. However, taking the worst case into account may end up with an extremely large headroom buffer which makes CaPFC unfeasible. Therefore, a statistical multiplexing approach is adopted since egress buffer overflow is not desirable but tolerable. The cost of egress overflow with very low probability is usually less than under utilization of buffer and link capacities. We tuned to the optimal value by simulations. XON threshold in egress buffers is set with the same idea in the ingress buffers.

The WARN threshold is set less than the XOFF threshold to have a fresh picture of the congestion contribution before asking the input ports to send pause messages. However, it should not be set less than the XON threshold because in persistent congestion states, the buffer occupancy may not drop well below the XON threshold. Therefore, the counters may be not be reset for a long time if the WARN threshold is set lower than the XON threshold. It results in expired statistics about the contributors of congestion. We set the WARN threshold the same value as the XON threshold at the egress buffers because it enables using the existing XON threshold in hardware for the purposes of the WARN threshold to simplify the implementation.

IV. Performance Evaluation

A. NS-3 Simulation Setup

The NS-3 simulation setup is built using the code base of DeTail [8] work to evaluate the performance of CaPFC compared to PFC. Even though it is missing in [8], the code base uses Network Simulation Cradle (NSC), a framework that embeds real Linux code in the NS-3 simulation. It enables to get more realistic results and it has ECN functionality. We also adopted the topology based on the input from Section 5.4 of [10] which is larger than the one in [8]. The simulated switch has a pipeline based packet processor as presented in Section II-A. It employs FIFO and ETS for ingress and egress scheduling, respectively. The parameters of this switch are taken from [8] which assumes 25μs switch delay. We vary the packet processing speed of the pipeline-based switch from $1M$ packets/s to $0.5M$ packets/s. ECN is implemented using NSC's TCP stack. ECN threshold is set to $20KB$ as optimized by the simulations. The adaptive load balancing thresholds explained in [8] are set to $16KB$ and $32KB$, respectively.

B. Traffic Characterization

In data center networks, traffic can be characterized by mainly three different traffic types namely real-time query traffic, latency-sensitive short messaging traffic and throughput-oriented long background traffic [3]. Query traffic is also sensitive to latency in addition to having a risk of throughput collapse named as TCP Incast [11]. We simulate all three traffic patterns concurrently inside the network. Inter-arrival times of the long traffic and short traffic flows are exponentially distributed with a mean of 5000 and end-nodes of the flows are uniformly selected. Query traffic consists of randomly selected 40 source nodes sending a fixed size of flow to a randomly selected common aggregation node simultaneously. The inter-arrival time of the query traffic is also exponentially distributed with the expected arrival rate of 100. These traffic types are differentiated from each other by setting different priorities. Some of these priorities may use Strict Priority (SP) Queues, whereas some others can go over Weighted Deficit Round Robin (WDDR) scheduling. We created six traffic simulation scenarios for the dedicated buffer memory and one for shared buffer memory setup to test the proposed techniques in different conditions. The traffic scenarios for the dedicated memory setup are given in Table I. PS is the packet processing speed of the pipeline in terms of packets per second.

TABLE I. Different Traffic Simulation Scenarios

		Long Traffic		Short Traffic		Query Traffic	
Scenario No	PS	Size	Queue Type	Size	Queue Type	Size	Queue Type
1	1Mpps	1MB	WDRR	64KB	SP	16KB	WDRR
2	1Mpps	1MB	WDRR	16KB	SP	16KB	WDRR
3	1Mpps	1MB	WDRR	64KB	SP	32KB	WDRR
4	1Mpps	1MB	WDRR	64KB	WDRR	16KB	SP
5	1Mpps	1MB	WDRR	64KB	WDRR	16KB	WDRR
6	0.5Mpps	1MB	WDRR	64KB	SP	16KB	WDRR

C. Simulation Results

In this section, we investigate the performance of two versions of CaPFC compared to two versions of PFC used in DeTail in terms of tail (maximum) flow completion time (FCT). CaPFC_max implements the Stop-Max algorithm whereas CaPFC_cal implements the Stop-Calibrate algorithm. In PFC with drop (PFC_wdrop), a packet is dropped at the end of pipeline if the destination egress queue has no space, whereas PFC with stop (PFC_wstop) pauses packet processing in that case to prevent any packet drop.

The test network is a Fat-Tree topology with 8-port 128 servers and 80 switches from [10]. Since the simulation employs NSC, it is not easy to try larger networks. The capacity of the links are $1Gbs$ and the packet processing speed is either $1M$ packets/s or $0.5M$ packets/s depending on the simulation scenario in Table I. Total subscription ratio is 4, two from ToR to aggregate and two from aggregate to core. The values of parameters are given in Table II. We note that MAX, XOFF, and XON thresholds are per-port per-priority.

TABLE II. Simulation Parameters

Parameter	Value	Parameter	Value
Pipeline speed	0.5Mpps or 1Mpps	Ingress XON thres.	40KB
Switch delay	25μs	Egress MAX thres.	60KB
ECN thresh.	20KB	Egress XOFF thres.	25KB
ALB thresh. (min/max)	16KB/32KB	Egress XON thres.	20KB
Ingress MAX thresh.	60KB	Egress WARN thres.	20KB
Ingress XOFF thres.	50KB	CUT thresh.	80%

We take the simulation scenario 1 as the base case and observe the effect of different parameters on the performance by changing them one-by-one in each scenario. In the base scenario, short message traffic has higher priority than the rest of the traffic. In Fig. 8(a), tail FCT of four different techniques are shown with a breakdown in terms of traffic type. The major observations are detailed in the following sections.

1) CaPFC provides high burst absorbtion for query traffic: Both versions of CaPFC reduces the maximum FCT of the query traffic approximately 6 times and 3 times compared to PFC_wdrop and PFC_wstop, respectively. CaPFC absorbed the traffic bursts in a much better way than both versions of PFC. Query traffic experiences TCP incast most severly

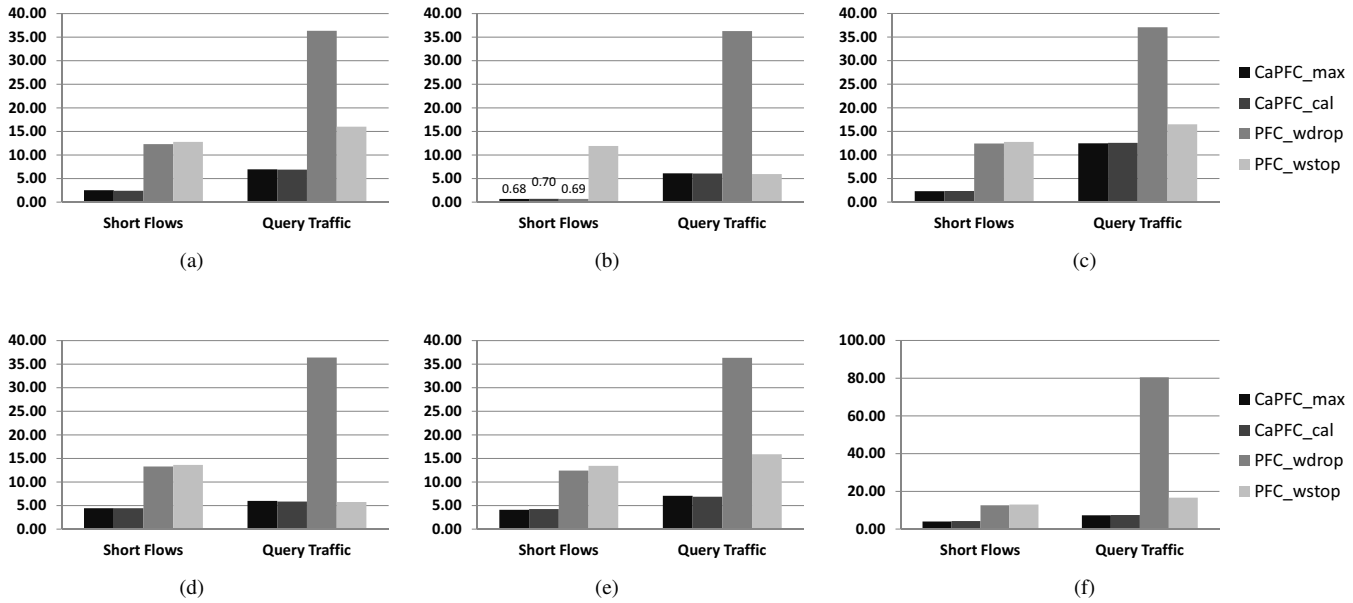

Fig. 8. Comparison of different versions of CaPFC and PFC in terms of tail FCT (unit of y-axis is miliseconds) in (a) simulation scenario 1, (b) simulation scenario 2, (c) simulation scenario 3 (d) simulation scenario 4, (e) simulation scenario 5, and (f) simulation scenario 6.

under PFC_wdrop mechanism since sudden bursts lead to synchronous packet drops.

2) CaPFC satisfies low latency for the short message flows: In terms of tail latency of the short message flows, both versions of CaPFC resulted in approximately 5 fold savings compared to both versions of PFC. PFC_wdrop suffers from fast retransmissions and timeouts whereas PFC_wstop suffers from HOL blocking and congestion spread.

In the rest of the evaluation, we change some of the simulation and traffic parameters one-by-one to see their effect on the performance of techniques. These scenarios are outlined in Table I from scenario 2 to 6.

3) CaPFC is efficient in different flow sizes: In scenario 2, the size of the short flows are decreased from $64KB$ to $16KB$. Similarly, in scenario 3, the size of the query traffic flows are doubled to $32KB$ compared to the base case. In Fig. 8(b) and Fig. 8(c), the tail FCT of the query and the short messaging traffic are shown for simulation scenario 2 and 3. Interestingly, in Fig. 8(b), PFC_wdrop incurs approximately 17 times more tail FCT for short flows whereas PFC_wstop incurs approximately 6 times more tail FCT for the query traffic. The results in Fig. 8(b) and Fig. 8(c) are consistent with the results in Fig. 8(a), which means both versions of CaPFC performs good for different flow sizes.

4) CaPFC is efficient in different prioritization configurations: In scenarios 4 and 5, we change the queue types of the short and query flows. Originally, short message traffic has the highest priority. In simulation scenario 4, the query traffic has the highest priority whereas in scenario 5, all traffic types go through separate WDRR queues. The results are presented in Fig. 8(d) and Fig. 8(e). These results are also consistent with the results in scenario 1.

5) CaPFC is efficient in different packet processing speeds: Lastly, we decrease the processing speed of the pipeline to half. It puts the pressure more on the ingress buffers therefore egress buffer occupancy becomes less important. According to the results in Fig. 8(f), changing the speed of the pipeline does not curb the advantage of CaPFC over PFC_wstop but amplifies over PFC_wdrop.

TABLE III. LONG BACKGROUND TRAFFIC TAIL FCT

	Tail FCT (ms) Based on Scenario					
Technique	Sce. 1	Sce. 2	Sce. 3	Sce. 4	Sce. 5	Sce. 6
CaPFC_max	52.56	51.08	55.13	58.55	50.87	58.32
CaPFC_cal	53.21	55.34	51.77	57.80	49.32	55.55
PFC_wdrop	50.67	49.80	62.42	56.78	51.57	56.08
PFC_wstop	55.22	55.87	52.17	53.46	61.02	59.75

6) CaPFC offers high throughput for the long background flows: The tail FCT of the long background flows are given in Table III for every scenario. Even though CaPFC resulted in significant savings in terms of tail latencies of the query traffic and short message traffic, it did not compromise the bandwidth requirement of the long background flows. Both versions of CaPFC had similar results compared to both versions of PFC. In the worst case scenario, CaPFC results in 4% increase in the tail FCT compared to PFC_wdrop in Scenario 6. It improves the tail FCT up to 11.7% and 16.6% compared to PFC_wdrop and PFC_wstop, respectively.

7) CaPFC is efficient in shared memory architectures: We modify the switch architecture to observe the performance of both versions of CaPFC in shared memory switches. Each priority queue has $8KB$ dedicated memory and there is total $350KB$ available memory in the shared pool. The maximum shared buffer capacity that can be used by each priority queue and by each port are $80KB$ and $100KB$, respectively. In the egress queues, we set the XOFF, XON, and WARN thresholds to $50KB$, $35KB$, and $35KB$, respectively. In the ingress queues, XOFF threshold is set to $30KB$ whereas XON threshold is set to $20KB$. The traffic scenario is given as scenario 1

in Table I. The tail FCT of query traffic and short messaging traffic are given in Fig. 9. Even though both versions of CaPFC perform significantly better than both versions of PFC for the query traffic, CaPFC_cal is 3 times better then CaPFC_max. Shared memory architecture helps both versions of PFC in terms of tail FCT of the short messaging traffic. However, PFC_wdrop doubles the tail FCT of short messaging traffic compared to both versions of CaPFC and PFC_wdrop.

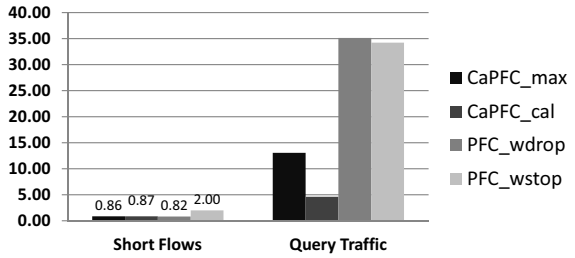

Fig. 9. Tail FCT (ms) results for all techniques in a shared memory switch.

V. Related Work

Flow control and congestion control are addressed in different layers and in different places in the network. Recently, limitations of PFC, such as HOL blocking and unfairness, are highlighted in [12]. Authors propose DCQCN a new congestion protocol based on QCN and DCTCP. DCQCN is an end-to-end protocol that involves higher layers. As an alternative, we propose a purely link-layer-based hop-by-hop solution. In [13], limitations of PFC are also listed as large buffering delays, unfairness, HOL blocking, and deadlock. Similarly, PFC is combined with DCTCP [3] and a deadlock-free routing scheme is proposed in [13]. In this paper, we assume no changes to TCP and propose a solution based on the link layer, which can react faster to flash congestion. A different approach is taken in [14], a modified QCN technique is proposed to reduce the side effects of PFC. It complements per-port per-priority granularity of PFC with the per-flow granularity of QCN. However, it inherits all three limitations of QCN described in Section I.

In [7], the impacts of PFC and QCN over different TCP mechanisms are investigated. The results suggest that PFC improves the TCP performance in every case and emphasize on the deficiencies of QCN as burst sensitivity, lack of adaptivity and unfairness. The performance of PFC over different traffic scenarios is investigated in [15], which elaborates the limitations of PFC, which are starvation and unfairness, arising due to interactions of different flows. Compared to these works, CaPFC improves the benefits of PFC by alleviating some of these limitations utilizing from both ingress and egress queue statistics to take proactive action earlier.

Infiniband uses a credit based flow control algorithm to ensure lossless service [16]. A receiver advertises to the sender the free capacity in its buffers and the sender does not exceed that limit while transmitting data. However, Infiband is much less ubiquitous than Ethernet. Virtual output queueing (VOQ) is also used to alleviate HOL blocking when there is an association between ingress and egress queues, which is not possible at the switch architecture we take reference in Section II-A. VOQ is a costly operation but more importantly, it does not differentiate between heavy and light flows. Therefore, it leads to long queueing delays for light short flows, when the egress queue is full.

Congestion management has been a popular topic in data center networks for some time. Most of the recently proposed techniques address this problem at the network or transport layer. The survey in [17] breaks down the techniques dealing with this problem into four subcategories, reducing queue length, accelerating retransmission, prioritizing mice flows, and exploiting multipath. DCTCP [3] is acknowledged to be a seminal work in this field. DeTail [8] is one of those techniques and is unique for utilizing both prioritization and multipath. It is also unique because it implements PFC to provide lossless link layer operation to complement the per-packet adaptive load balancing algorithm that reduces tail flow completion times in data center networks. However, in [8], the main switching mechanism is characterized as a crossbar rather than a pipeline mechanism, which is taken as the main reference in this paper. Also in [8], analysis about different switch types is shallow.

Finally, the lack of fairness in QCN is pointed out by [11] and [18] and its effect on the network performance for the TCP incast scenarios are shown and a modified version of QCN is proposed, namely FQCN in [11], to mitigate this drawback. [18] also proposes a modification to QCN, namely AF-QCN, which provides faster fairness convergence and enables weighted fairness. Even though these modifications improve the performance of QCN, they could not help it to gain widespread adoption.

VI. Conclusion

PFC and QCN are proposed as parts of DCB to enhance the capabilities of Ethernet. However, they have certain limitations that bars their wide deployment. First, PFC only relies on monitoring ingress queues, therefore, it is not protected against saturation trees and HOL. Second, QCN has multiple issues but most importantly it cannot pass VLAN borders. As a remedy to the shortcomings of PFC and QCN, we propose CaPFC as a congestion aware flow control mechanism. CaPFC monitors both ingress and egress queues and takes proactive action to prevent egress buffer overflow which causes congestion spread and HOL blocking.

By implementing CaPFC, we achieve four features of low latency data center networks. First, we use prioritization and use SP queueing for latency-sensitive flows. Also, we provide parallel lanes for each type of traffic by preventing congestion in one type of traffic to spread to other types of flows. Second, by proactively monitoring both ingress and egress queues, we minimize HOL blocking and retransmissions due to packet drops. This in turn increases link utilization and reduces average queue length, which also reduces latency.

We carried out simulations in NS-3 to evaluate the performance of CaPFC compared to PFC. The results show that CaPFC is able to reduce maximum FCT of the query traffic up to 6 times. It also reduces the maximum FCT of the short messaging traffic up to 17 times. In addition, CaPFC keeps the performance improvement on various traffic scenarios and switch parameters. On the other hand, it does not sacrifice the bandwidth requirement of the long background traffic at

the expense of dramatic performance of the latency-sensitive traffic. For future work, we plan to implement CaPFC with different transport protocols, such as DCTCP, and different load balancing schemes, such as [10].

As a future work, we also consider to test CaPFC in a realistic topology with real network loads to observe tighter queue limits as the number of ports in the switch and the number of senders in query traffic increase. We expect that CaPFC would still provide low latency for short and query traffic flows at the cost of little bandwidth loss for large flows.

REFERENCES

[1] K. Won, "Trends reshaping networks," http://www.networkworld.com/article/2198766/tech-primers/trends-reshaping-networks.html, 2011.

[2] D. Crisan, "Optimized protocol stack for virtualized converged enhanced ethernet," Ph.D. dissertation, ETH Zurich, 2014.

[3] M. Alizadeh et. al., "Data center TCP (DCTCP)," in *Proc. SIGCOMM '10*, vol. 1, New York, 2010, pp. 63–74.

[4] "Quantized congestion notification and todays fibre channel over ethernet networks," Cisco Systems, Inc, Cisco Website, Tech. Rep., 2014.

[5] "High capacity StrataXGS® Trident II ethernet switch series," http://www.broadcom.com/products/Switching/Data-Center/BCM56850-Series, 2014.

[6] M. Gusat, C. Minkenberg, and G. J. Paljack, "Flow and congestion control for datacenter networks," IBM, Tech. Rep., 2009.

[7] D. Crisan et. al., "Short and fat: Tcp performance in CEE datacenter networks," in *Proc. HOTI 2011*, Santa Clara, CA, August 2011.

[8] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: reducing the flow completion time tail in datacenter networks," in *Proc. SIGCOMM '12*, vol. 1, New Delhi, 2012, pp. 139–150.

[9] "Priority flow control: Build reliable layer 2 infrastructure," Cisco Systems, Inc, Cisco Website, Tech. Rep., 2009.

[10] J. Cao et. al., "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proc. CoNEXT '13*, Santa Barbara, CA, December 2013.

[11] Y. Zhang and N. Ansari, "On mitigating TCP incast in data center networks," in *Proc. INFOCOM 2011*, Shangai, April 2011.

[12] Y. Z. et. al., "Congestion control for large-scale RDMA deployments," in *Proc. SIGCOMM '15*, vol. 1, London, 2015, pp. 523–536.

[13] B. Stephens et. al., "Practical DCB for improved data center networks," in *Proc. IEEE INFOCOM 2014*, Toronto, April 2014, pp. 1824–1832.

[14] F. D. Neeser et. al., "Occupancy sampling for terabit CEE switches," in *Proc. HOTI 2012*, Santa Clara, CA, August 2012.

[15] M. Haggen and R. Zarick, "Performance evaluation of DCBs priority-based flow control," in *Proc.* 10^{th} *International Symposium on Network Computing Applications (NCA)*, Cambridge, MA, August 2011.

[16] "Infiniband FAQ, rev 1.3," Mellanox Technologies, Mellanox Website, Tech. Rep., December 2014.

[17] S. Liu, H. Xu, and Z. Cai, "Low latency datacenter networking: A short survey," *http://arxiv.org/abs/1312.3455*, 2014.

[18] A. Kabbani et. al., "Approximate fairness with quantized congestion notification for multi-tenanted data centers," in *Proc. HOTI 2010*, Mountain View, CA, August 2010.